We introduce Stanza, an open-source Python natural language processing toolkit supporting 66 human languages. Compared to existing widely used toolkits, Stanza features a language-agnostic fully neural pipeline for text analysis, including tokenization, multi-word token expansion, lemmatization, part-of-speech and morphological feature tagging, dependency parsing, and named entity recognition. We have trained Stanza on a total of 112 datasets, including the Universal Dependencies treebanks and other multilingual corpora, and show that the same neural architecture generalizes well and achieves competitive performance on all languages tested. Additionally, Stanza includes a native Python interface to the widely used Java Stanford CoreNLP software, which further extends its functionality to cover other tasks such as coreference resolution and relation extraction. Source code, documentation, and pretrained models for 66 languages are available at https://stanfordnlp.github.io/stanza/.

1 Introduction

The growing availability of open-source natural language processing (NLP) toolkits has made it easier for users to build tools with sophisticated linguistic processing. While existing NLP toolkits such as CoreNLP (Manning et al., 2014), FLAIR (Akbik et al., 2019), spaCy\(^1\), and UDPipe (Straka, 2018) have had wide usage, they also suffer from several limitations. First, existing toolkits often support only a few major languages. This has significantly limited the community’s ability to process multilingual text. Second, widely used tools are sometimes under-optimized for accuracy either due to a focus on efficiency (e.g., spaCy) or use of less powerful models (e.g., CoreNLP), potentially mislead-

\(^1\)https://spacy.io/

\(^2\)The toolkit was called StanfordNLP prior to v1.0.0.
Table 1: Feature comparisons of Stanza against other popular natural language processing toolkits.

<table>
<thead>
<tr>
<th>System</th>
<th># Human Languages</th>
<th>Programming Language</th>
<th>Raw Text Processing</th>
<th>Fully Neural</th>
<th>Pretrained Models</th>
<th>State-of-the-art Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoreNLP</td>
<td>6</td>
<td>Java</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>FLAIR</td>
<td>12</td>
<td>Python</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>spaCy</td>
<td>10</td>
<td>Python</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>UDPipe</td>
<td>61</td>
<td>C++</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Stanza</td>
<td>66</td>
<td>Python</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

**State-of-the-art performance.** We evaluate Stanza on a total of 112 datasets, and find its neural pipeline adapts well to text of different genres, achieving state-of-the-art or competitive performance at each step of the pipeline.

Additionally, Stanza features a Python interface to the widely used Java CoreNLP package, allowing access to additional tools such as coreference resolution and relation extraction.

Stanza is fully open source and we make pretrained models for all supported languages and datasets available for public download. We hope Stanza can facilitate multilingual NLP research and applications, and drive future research that produces insights from human languages.

2 System Design and Architecture

At the top level, Stanza consists of two individual components: (1) a fully neural multilingual NLP pipeline; (2) a Python client interface to the Java Stanford CoreNLP software. In this section we introduce their designs.

2.1 Neural Multilingual NLP Pipeline

Stanza’s neural pipeline consists of models that range from tokenizing raw text to performing syntactic analysis on entire sentences (see Figure 1). All components are designed with processing many human languages in mind, with high-level design choices capturing common phenomena in many languages and data-driven models that learn the difference between these languages from data. Moreover, the implementation of Stanza components is highly modular, and reuses basic model architectures when possible for compactness. We highlight the important design choices here, and refer the reader to Qi et al. (2018) for modeling details.

Tokenization and Sentence Splitting. When presented raw text, Stanza tokenizes it and groups tokens into sentences as the first step of processing. Unlike most existing toolkits, Stanza combines tokenization and sentence segmentation from raw text into a single module. This is modeled as a tagging problem over character sequences, where the model predicts whether a given character is the end of a token, end of a sentence, or end of a multi-word token (MWT, see Figure 2). We choose to predict MWTs jointly with tokenization because this task is context-sensitive in some languages.

Multi-word Token Expansion. Once MWTs are identified by the tokenizer, they are expanded into the underlying syntactic words as the basis of downstream processing. This is achieved with an ensemble of a frequency lexicon and a neural sequence-to-sequence (seq2seq) model, to ensure that frequently observed expansions in the training set are always robustly expanded while maintaining flexibility to model unseen words statistically.

POS and Morphological Feature Tagging. For each word in a sentence, Stanza assigns it a part-of-speech (POS), and analyzes its universal morphological features (UFeats, e.g., singular/plural, 1st/2nd/3rd person, etc.). To predict POS and UFeats, we adopt a bidirectional long short-term memory network (Bi-LSTM) as the basic architecture. For consistency among universal POS (UPOS),

Following Universal Dependencies (Nivre et al., 2020), we make a distinction between tokens (contiguous spans of characters in the input text) and syntactic words. These are interchangeable aside from the cases of MWTs, where one token can correspond to multiple words.
treebank-specific POS (XPOS), and UFeats, we adopt the biaffine scoring mechanism from Dozat and Manning (2017) to condition XPOS and UFeats prediction on that of UPOS.

**Lemmatization.** Stanza also lemmatizes each word in a sentence to recover its canonical form (e.g., did→do). Similar to the multi-word token expander, Stanza’s lemmatizer is implemented as an ensemble of a dictionary-based lemmatizer and a neural seq2seq lemmatizer. An additional classifier is built on the encoder output of the seq2seq model, to predict shortcuts such as lowercasing and identity copy for robustness on long input sequences such as URLs.

**Dependency Parsing.** Stanza parses each sentence for its syntactic structure, where each word in the sentence is assigned a syntactic head that is either another word in the sentence, or in the case of the root word, an artificial root symbol. We implement a Bi-LSTM-based deep biaffine neural dependency parser (Dozat and Manning, 2017). We further augment this model with two linguistically motivated features: one that predicts the linearization order of two words in a given language, and the other that predicts the typical distance in linear order between them. We have previously shown that these features significantly improve parsing accuracy (Qi et al., 2018).

**Named Entity Recognition.** For each input sentence, Stanza also recognizes named entities in it (e.g., person names, organizations, etc.). For NER we adopt the contextualized string representation-based sequence tagger from Akbik et al. (2018). We first train a forward and a backward character-level LSTM language model, and at tagging time we concatenate the representations at the end of each word position from both language models with word embeddings, and feed the result into a standard one-layer Bi-LSTM sequence tagger with a conditional random field (CRF)-based decoder.

### 2.2 CoreNLP Client

Stanford’s Java CoreNLP software provides a comprehensive set of NLP tools especially for the English language. However, these tools are not easily accessible with Python, the programming language of choice for many NLP practitioners, due to the lack of official support. To facilitate the use of CoreNLP from Python, we take advantage of the existing server interface in CoreNLP, and implement a robust client as its Python interface.

When the CoreNLP client is instantiated, Stanza will automatically start the CoreNLP server as a local process. The client then communicates with the server through its RESTful APIs, after which annotations are transmitted in Protocol Buffers, and converted back to native Python objects. Users can also specify JSON or XML as annotation format. To ensure robustness, while the client is being used, Stanza periodically checks the health of the server, and restarts it if necessary.

### 3 System Usage

Stanza’s user interface is designed to allow quick out-of-the-box processing of multilingual text. To achieve this, Stanza supports automated model download via Python code and pipeline customization with processors of choice. Annotation results can be accessed as native Python objects to allow for flexible post-processing.

#### 3.1 Neural Pipeline Interface

Stanza’s neural NLP pipeline can be initialized with the Pipeline class, taking language name as an argument. By default, all processors will be loaded and run over the input text; however, users can also specify the processors to load and run with a list of processor names as an argument. Users can additionally specify other processor-level properties, such as batch sizes used by processors, at initialization time.

The following code snippet shows a minimal usage of Stanza for downloading the Chinese model, annotating a sentence with customized processors, and printing out all annotations:

```python
import stanza
# download Chinese model
stanza.download('zh')
# initialize Chinese neural pipeline
nlp = stanza.Pipeline('zh', processors='tokenize,pos,ner')
# run annotation over a sentence
doc = nlp('斯坦福是一所私立研究型大学。')
print(doc)
```

After all processors are run, a Document instance will be returned, which stores all annotation results. Within a Document, annotations are further stored in Sentences, Tokens and Words in a top-down fashion (Figure 1). The following code snippet demonstrates how to access the text and POS tag of each word in a document and all named entities in the document:
Stanza is designed to be run on different hardware devices. By default, CUDA devices will be used whenever they are visible by the pipeline, or otherwise CPUs will be used. However, users can force all computation to be run on CPUs by setting `use_gpu=False` at initialization time.

### 3.2 CoreNLP Client Interface

The CoreNLP client interface is designed in a way that the actual communication with the backend CoreNLP server is transparent to the user. To annotate an input text with the CoreNLP client, a `CoreNLPClient` instance needs to be initialized, with an optional list of CoreNLP annotators. After the annotation is complete, results will be accessible as native Python objects.

This code snippet shows how to establish a CoreNLP client and obtain the NER and coreference annotations of an English sentence:

```python
from stanza.server import CoreNLPClient
# start a CoreNLP client
with CoreNLPClient(annotators=['tokenize','ssplit','pos','lemma','ner','parse','coref']) as client:
    # run annotation over input
    ann = client.annotate('Emily said that she liked the movie.
    # access all entities
    for sent in ann.sentences:
        print(sent.mentions)
    # access coreference annotations
    print(ann.corefChain)
```

With the client interface, users can annotate text in 6 languages as supported by CoreNLP.

### 3.3 Interactive Web-based Demo

To help visualize documents and their annotations generated by Stanza, we build an interactive web demo that runs the pipeline interactively. For all languages and all annotations Stanza provides in those languages, we generate predictions from the models trained on the largest treebank/NER dataset, and visualize the result with the Brat rapid annotation tool. This demo runs in a client/server architecture, and annotation is performed on the server side. We make one instance of this demo publicly available at [http://stanza.run/](http://stanza.run/). It can also be run locally with proper Python libraries installed.

### 4 Performance Evaluation

To establish benchmark results and compare with other popular toolkits, we trained and evaluated Stanza on a total of 112 datasets. All pretrained models are publicly downloadable.

**Datasets.** We train and evaluate Stanza’s tokenizer/sentence splitter, MWT expander, POS/UFeats tagger, lemmatizer, and dependency parser with the Universal Dependencies v2.5 treebanks (Zeman et al., 2019). For training we use 100 treebanks from this release that have non-copyrighted training data, and for treebanks that do not include development data, we randomly split out 20% of
<table>
<thead>
<tr>
<th>Treebank</th>
<th>System</th>
<th>Tokens</th>
<th>Sents.</th>
<th>Words</th>
<th>UPOS</th>
<th>XPOS</th>
<th>UFeats</th>
<th>Lemmas</th>
<th>UAS</th>
<th>LAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall (100 treebanks)</td>
<td>Stanza</td>
<td>99.09</td>
<td>86.05</td>
<td>98.63</td>
<td>92.49</td>
<td>91.80</td>
<td>89.93</td>
<td>92.78</td>
<td>80.45</td>
<td>75.68</td>
</tr>
<tr>
<td>Arabic-PADT</td>
<td>Stanza</td>
<td>99.98</td>
<td>80.43</td>
<td>97.88</td>
<td>94.89</td>
<td>91.75</td>
<td>91.86</td>
<td>93.27</td>
<td>83.27</td>
<td>79.33</td>
</tr>
<tr>
<td></td>
<td>UDPipe</td>
<td>99.98</td>
<td>82.09</td>
<td>94.58</td>
<td>90.36</td>
<td>84.00</td>
<td>84.16</td>
<td>88.46</td>
<td>72.67</td>
<td>68.14</td>
</tr>
<tr>
<td>Chinese-GSD</td>
<td>Stanza</td>
<td>92.83</td>
<td>98.80</td>
<td>92.83</td>
<td>89.12</td>
<td>88.93</td>
<td>92.11</td>
<td>92.83</td>
<td>72.88</td>
<td>69.82</td>
</tr>
<tr>
<td></td>
<td>UDPipe</td>
<td>90.27</td>
<td>99.10</td>
<td>90.27</td>
<td>84.13</td>
<td>84.04</td>
<td>89.05</td>
<td>90.26</td>
<td>61.60</td>
<td>57.81</td>
</tr>
<tr>
<td>English-EWT</td>
<td>Stanza</td>
<td>99.01</td>
<td>81.13</td>
<td>99.01</td>
<td>95.40</td>
<td>95.12</td>
<td>96.11</td>
<td>97.21</td>
<td>86.22</td>
<td>73.59</td>
</tr>
<tr>
<td></td>
<td>UDPipe</td>
<td>98.90</td>
<td>77.40</td>
<td>98.90</td>
<td>93.26</td>
<td>92.75</td>
<td>94.23</td>
<td>95.45</td>
<td>80.22</td>
<td>77.03</td>
</tr>
<tr>
<td></td>
<td>spaCy</td>
<td>97.30</td>
<td>61.19</td>
<td>97.30</td>
<td>86.72</td>
<td>90.83</td>
<td>–</td>
<td>87.05</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>French-GSD</td>
<td>Stanza</td>
<td>99.68</td>
<td>94.92</td>
<td>99.48</td>
<td>97.30</td>
<td>–</td>
<td>96.72</td>
<td>97.64</td>
<td>91.38</td>
<td>89.05</td>
</tr>
<tr>
<td></td>
<td>UDPipe</td>
<td>99.68</td>
<td>93.59</td>
<td>98.81</td>
<td>95.85</td>
<td>–</td>
<td>95.55</td>
<td>96.61</td>
<td>87.14</td>
<td>84.26</td>
</tr>
<tr>
<td></td>
<td>spaCy</td>
<td>98.34</td>
<td>77.30</td>
<td>94.15</td>
<td>86.82</td>
<td>–</td>
<td>–</td>
<td>87.29</td>
<td>67.46</td>
<td>60.60</td>
</tr>
<tr>
<td>Spanish-AnCora</td>
<td>Stanza</td>
<td>99.98</td>
<td>99.07</td>
<td>99.98</td>
<td>98.78</td>
<td>98.67</td>
<td>98.59</td>
<td>99.19</td>
<td>92.21</td>
<td>90.01</td>
</tr>
<tr>
<td></td>
<td>UDPipe</td>
<td>99.97</td>
<td>98.32</td>
<td>99.95</td>
<td>98.32</td>
<td>98.13</td>
<td>98.13</td>
<td>98.48</td>
<td>88.22</td>
<td>85.10</td>
</tr>
<tr>
<td></td>
<td>spaCy</td>
<td>99.47</td>
<td>97.59</td>
<td>98.95</td>
<td>94.04</td>
<td>–</td>
<td>–</td>
<td>79.63</td>
<td>86.63</td>
<td>84.13</td>
</tr>
</tbody>
</table>

Table 2: Neural pipeline performance comparisons on the Universal Dependencies (v2.5) test treebanks. For our system we show macro-averaged results over all 100 treebanks. We also compare our system against UDPipe and spaCy on treebanks of five major languages where the corresponding pretrained models are publicly available. All results are F₁ scores produced by the 2018 UD Shared Task official evaluation script.

The training data as development data. These treebanks represent 66 languages, mostly European languages, but spanning a diversity of language families, including Indo-European, Afro-Asiatic, Uralic, Turkic, Sino-Tibetan, etc. For NER, we train and evaluate Stanza with 12 publicly available datasets covering 8 major languages as shown in Table 3 (Nothman et al., 2013; Tjong Kim Sang and De Meulder, 2003; Tjong Kim Sang, 2002; Benikova et al., 2014; Mohit et al., 2012; Taulé et al., 2008; Weischedel et al., 2013). For the WikiNER corpora, as canonical splits are not available, we randomly split them into 70% training, 15% dev and 15% test splits. For all other corpora we used their canonical splits.

**Training.** On the Universal Dependencies treebanks, we tuned all hyper-parameters on several large treebanks and applied them to all other treebanks. We used the word2vec embeddings released as part of the 2018 UD Shared Task (Zeman et al., 2018), or the fastText embeddings (Bojanowski et al., 2017) whenever word2vec is not available. For the character-level language models in the NER component, we pretrained them on a mix of the Common Crawl and Wikipedia dumps, and the news corpora released by the WMT19 Shared Task (Barrault et al., 2019), except for English and Chinese, for which we pretrained on the Google One Billion Word (Chelba et al., 2013) and the Chinese Gigaword corpora, respectively. We again applied the same hyper-parameters to models for all languages.

**Universal Dependencies Results.** For performance on UD treebanks, we compared Stanza (v1.0) against UDPipe (v1.2) and spaCy (v2.2) on treebanks of 5 major languages whenever a pretrained model is available. As shown in Table 2, Stanza achieved the best performance on most scores reported. Notably, we find that Stanza’s language-agnostic architecture is able to adapt to datasets of different languages and genres. This is also shown by Stanza’s high macro-averaged scores over 100 treebanks covering 66 languages.

**NER Results.** For performance of the NER component, we compared Stanza (v1.0) against FLAIR (v0.4.5) and spaCy (v2.2). For spaCy we reported results from its publicly available pretrained model whenever one trained on the same dataset can be found, otherwise we retrained its model on our datasets with default hyper-parameters, following the publicly available tutorial. For FLAIR, since their downloadable models were pretrained

---

5 https://catalog.ldc.upenn.edu/LDC2011T13
6 https://spacy.io/usage/training#ner

Note that, following this public tutorial, we did not use pretrained word embeddings when training spaCy NER models, although using pretrained word embeddings may potentially improve the NER results.
## Task

<table>
<thead>
<tr>
<th>Language</th>
<th>Corpus</th>
<th># Types</th>
<th>Stanza</th>
<th>FLAIR</th>
<th>spaCy</th>
<th>UDPipe CPU</th>
<th>FLAIR CPU</th>
<th>FLAIR GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>AQMAR</td>
<td>4</td>
<td>74.3</td>
<td>74.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Chinese</td>
<td>OntoNotes</td>
<td>18</td>
<td>79.2</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Dutch</td>
<td>CoNLL02</td>
<td>4</td>
<td>89.2</td>
<td>90.3</td>
<td>73.8</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>WikiNER</td>
<td>4</td>
<td>94.8</td>
<td>94.8</td>
<td>90.9</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>English</td>
<td>CoNLL03</td>
<td>4</td>
<td>92.1</td>
<td>92.7</td>
<td>81.0</td>
<td>88.8</td>
<td>89.0</td>
<td>85.4*</td>
</tr>
<tr>
<td></td>
<td>CoNLL04</td>
<td>18</td>
<td>92.9</td>
<td>92.5</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>French</td>
<td>WikiNER</td>
<td>4</td>
<td>92.9</td>
<td>92.5</td>
<td>88.8*</td>
<td>88.8</td>
<td>88.8</td>
<td>88.8*</td>
</tr>
<tr>
<td>German</td>
<td>CoNLL03</td>
<td>4</td>
<td>81.9</td>
<td>82.5</td>
<td>63.9</td>
<td>85.2</td>
<td>85.4</td>
<td>68.4</td>
</tr>
<tr>
<td></td>
<td>GermEval14</td>
<td>4</td>
<td>85.2</td>
<td>85.4</td>
<td>68.4</td>
<td>85.2</td>
<td>85.4</td>
<td>68.4</td>
</tr>
<tr>
<td>Russian</td>
<td>WikiNER</td>
<td>4</td>
<td>92.9</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Spanish</td>
<td>CoNLL02</td>
<td>4</td>
<td>88.1</td>
<td>87.3</td>
<td>77.5</td>
<td>86.6</td>
<td>86.4</td>
<td>76.1</td>
</tr>
<tr>
<td></td>
<td>AnCora</td>
<td>4</td>
<td>88.6</td>
<td>88.4</td>
<td>76.1</td>
<td>86.6</td>
<td>86.4</td>
<td>76.1</td>
</tr>
</tbody>
</table>

Table 3: NER performance across different languages and corpora. All scores reported are entity micro-averaged test F\(_1\). For each corpus we also list the number of entity types. * marks results from publicly available pretrained models on the same dataset, while others are from models retrained on our datasets.

on dataset versions different from canonical ones, we retrained all models on our own dataset splits with their best reported hyper-parameters. All test results are shown in Table 3. We find that on all datasets Stanza achieved either higher or close F\(_1\) scores when compared against FLAIR. When compared to spaCy, Stanza’s NER performance is much better. It is worth noting that Stanza’s high performance is achieved with much smaller models compared with FLAIR (up to 75% smaller), as we intentionally compressed the models for memory efficiency and ease of distribution.

### Speed comparison.

We compare Stanza against existing toolkits to evaluate the time it takes to annotate text (see Table 4). For GPU tests we use a single NVIDIA Titan RTX card. Unsurprisingly, Stanza’s extensive use of accurate neural models makes it take significantly longer than spaCy to annotate text, but it is still competitive when compared against toolkits of similar accuracy, especially with the help of GPU acceleration.

## 5 Conclusion and Future Work

We introduced Stanza, a Python natural language processing toolkit supporting many human languages. We have showed that Stanza’s neural pipeline not only has wide coverage of human languages, but also is accurate on all tasks, thanks to its language-agnostic, fully neural architectural design. Simultaneously, Stanza’s CoreNLP client extends its functionality with additional NLP tools.

### Acknowledgments

The authors would like to thank the anonymous reviewers for their comments, Arun Chaganty for his early contribution to this toolkit, Tim Dozat for his design of the original architectures of the tagger and parser models, Matthew Honnibal and Ines Montani for their help with spaCy integration and helpful comments on the draft, Ranting Guo for the logo design, and John Bauer and the community contributors for their help with maintaining and improving this toolkit. This research is funded in part by Samsung Electronics Co., Ltd. and in part by the SAIL-JD Research Initiative.
References


Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In Association for Computational Linguistics (ACL) System Demonstrations.


